Systèmes d'exploitation

Descripteurs de fichiers

Guido van Rossum

- Néerlandaise
- Université de Amsterdam
- Auteur de langage Python
- Dropbox

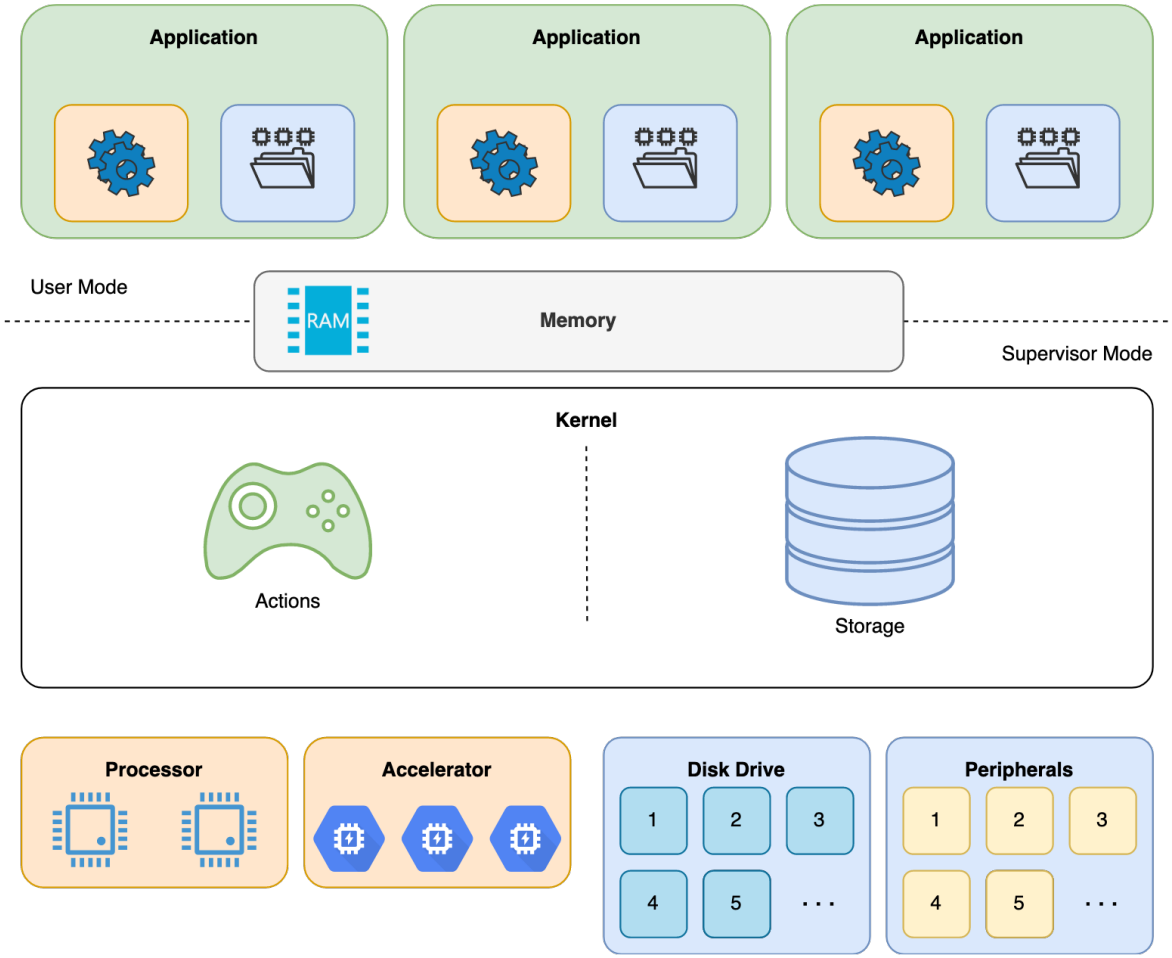
Contenu

- Appel de système
- Partition
- Fichier
- Directeur
- Fichiers Spéciaux

Bibliographie pour aujourd'hui

- Linux System Programming
  - Chapitre: File I/O
    - Opening Files / Closing Files
    - Reading via read() / Writing via write()
    - Seeking via lseek
    - Truncating Files
    - Kernel Internals
- Beginning Linux Programming
  - Chapitre 3
    - Linux File Structure
    - System Calls and Device Drivers
    - Library Functions
    - Low-Level File Access

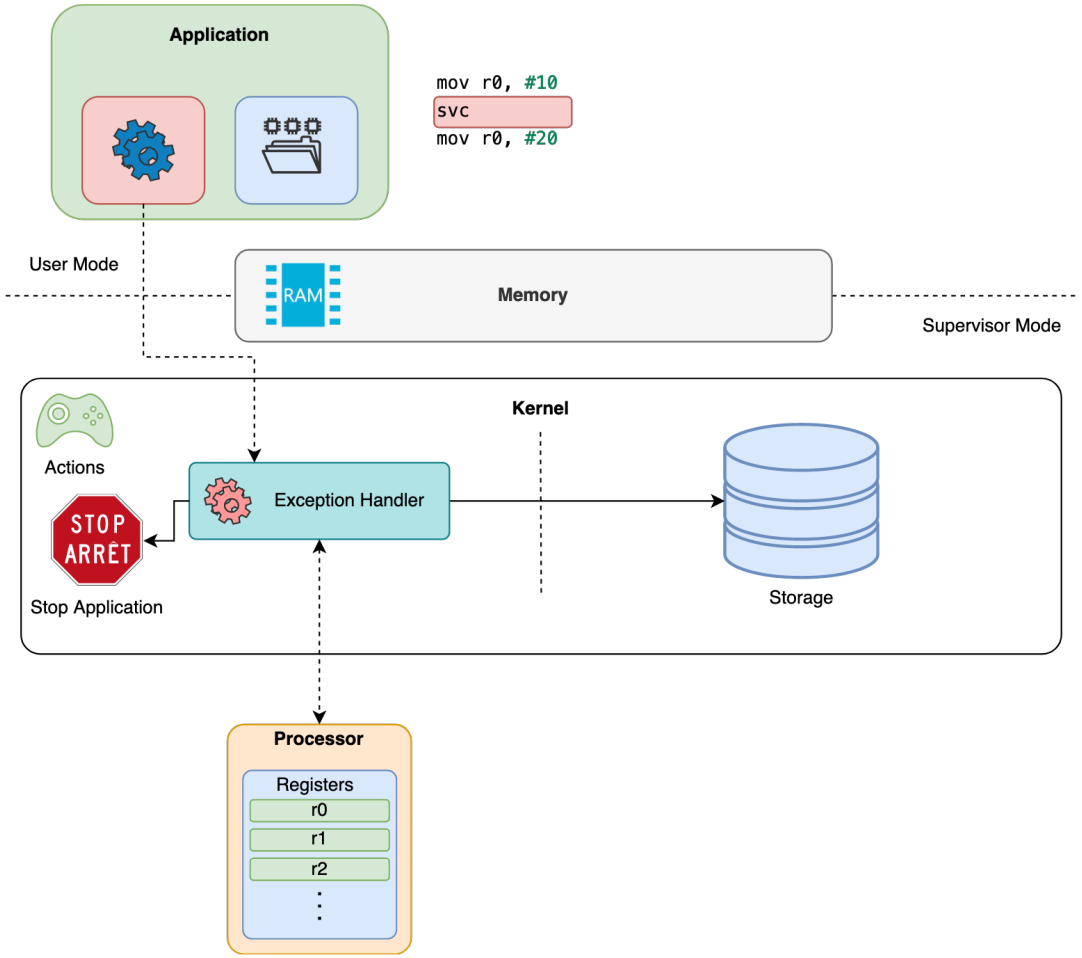
Idée General



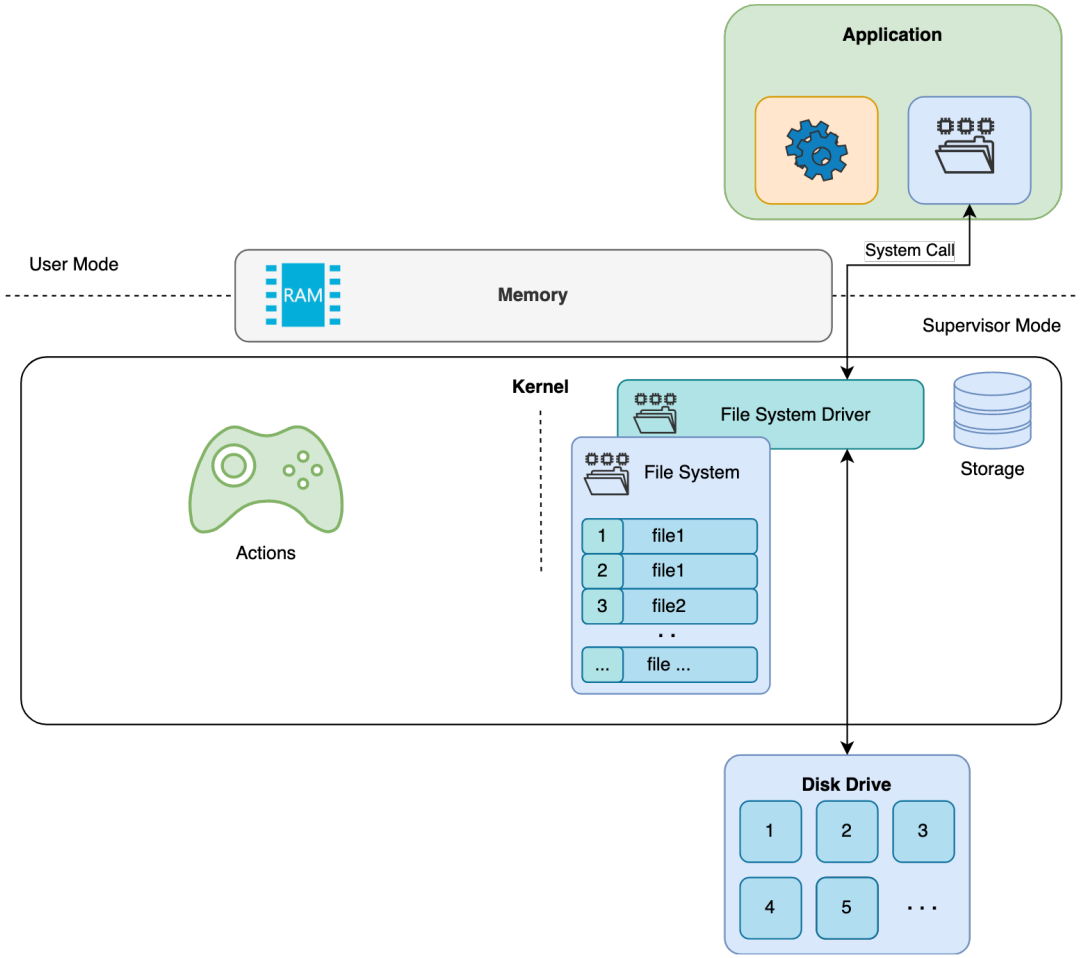
Modes d'exécution

- Le processeur a deux modes d'exécution
- **Mode superviseur**
  - Actions privilégiées
  - C'est le mode utilisé par le SE pour s'exécuter
- **Mode utilisateur**
  - L'accès direct au matériel n'est pas autorisé
  - L'espace adresse mémoire ne peut pas être modifié
  - C'est le mode utilisé pour les applications
- Le noyau este le intermédiaire pour:
  - l'accès des processus au matériel
  - l'accès des processus aux ressources
- **Appel système**
  - la transition du mode utilisateur au mode noyau

Appel du système

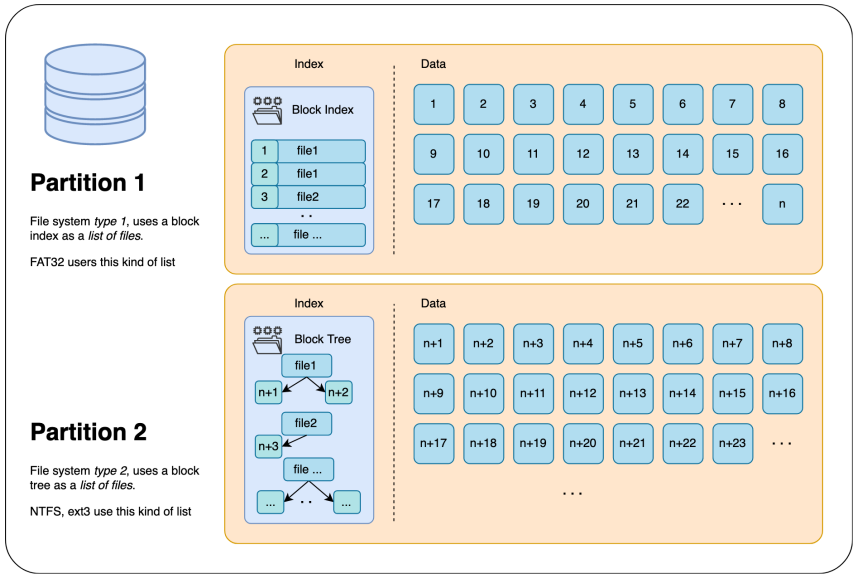


System de fichiers



System de fichiers

- transforme le tableau de blocs en fichiers et dossiers
- structures de données utilisées par le système d'exploitation



FICHIER

Types de fichiers

- fichiers simple – regular files
- directeurs – directories
- lien (symbolique) – hard/soft links
- dispositifs de type caractère – char devices
- dispositifs de type bloc – block devices
- pipes/FIFOS
- sockets UNIX
- Comment trouvez-vous le type d'un fichier sous UNIX?

OPERATIONS AVEC LES FICHIERS

Operations avec les fichiers

- création
- ouverture
- lecture
- écriture
- positionnement dans le fichier
- tronquer
- fermeture
- suppression

Creation

- **shell**

  touch /path/to/file

  toute commande qui écrit dans un fichier le crée

- **Rust**

```rust
let file = File::options()
    .read()
    .write()
    .create()
    .open("/path/to/file")
```

- **POSIX**

```c
int fd = open ("/path/to/file", O_CREAT | O_EXCL, 0644);
```

Ouverture

- **Rust**

```
let file = File::options()
    .read()
    .open("/path/to/file")
```

- **POSIX**

```
int fd = open("/path/to/file",
O_RDONLY);
```

Fichier Ouvert

- identification
  - Windows: poignée
  - POSIX: descripteur
  - pourquoi pas un nom?
- position
  - Windows: pointeur de fichier
  - Linux: curseur de fichier
- droits d'ouverture (mode)
- compteur d'utilisation (file-open count)

Descripteur de fichier

- plus de descripteur peut correspondre aux même fichier
- chaque processus a une table de descripteurs de fichiers

- **Rust**
  - struct File (a un descripteur de fichier à l'intérieur)

- **POSIX**
  - un entier identifiant une instance de fichier ouvert dans un processus
  - descripteurs spéciaux
    - 0 (stdin), 1 (stdout), 2 (stderr)

Tableau de descripteur de fichier

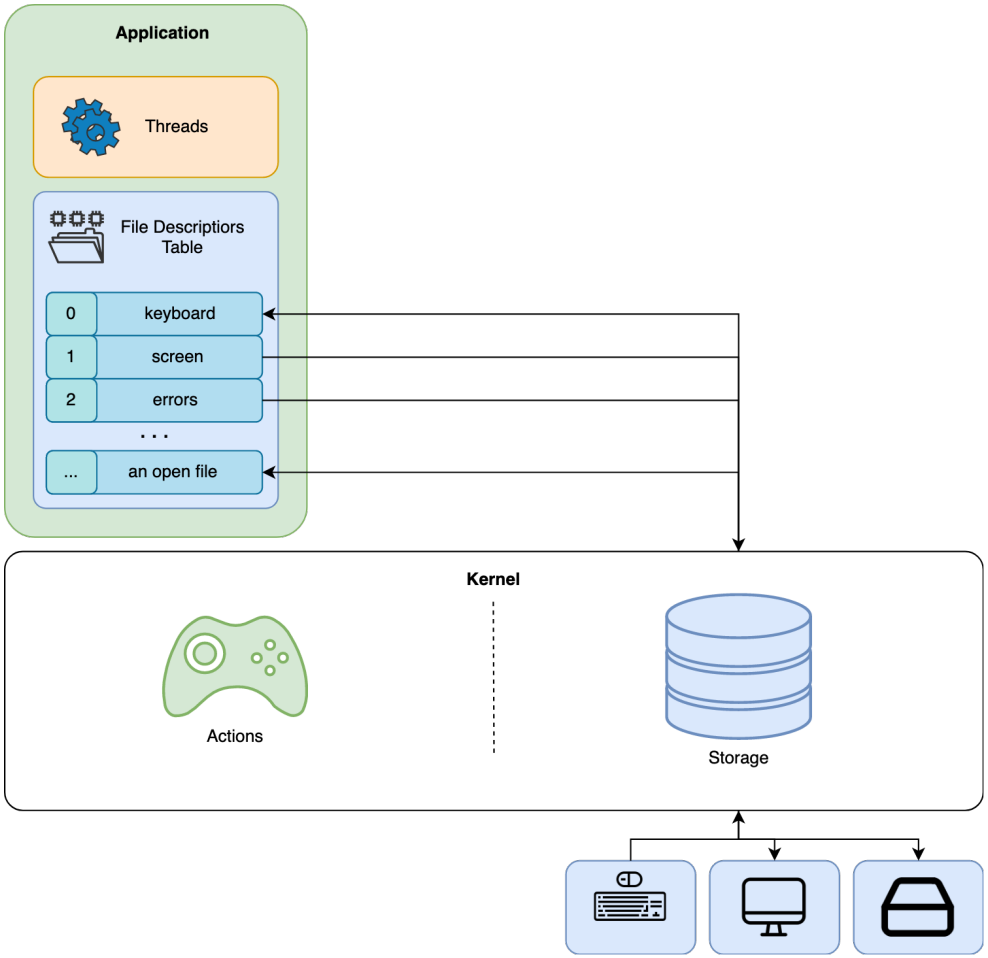
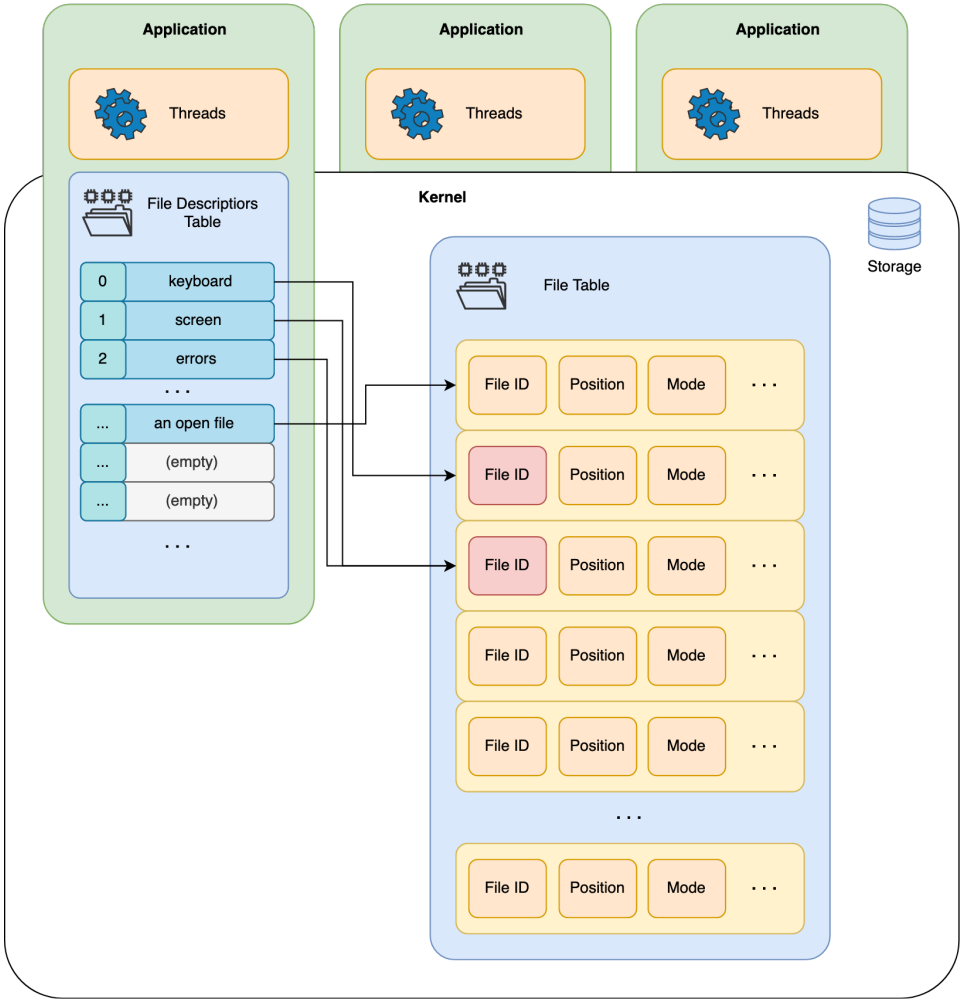
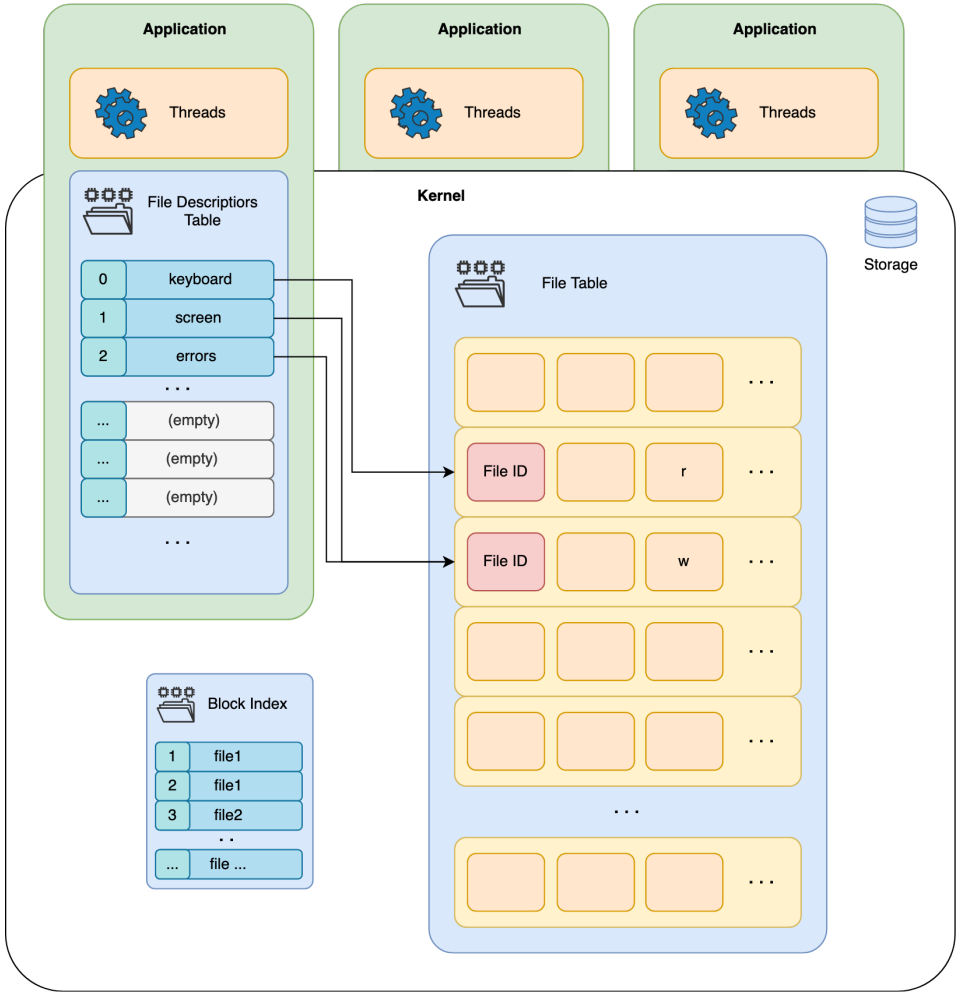


Tableau des fichiers



Ouverture


Application
Threads
Application
Threads
Application
Threads
Kernel
File Descriptiors Table
0
keyboard
1
screen
2
errors
...
(empty)
...
(empty)
...
(empty)
File Table
File ID
r
File ID
w
Storage
Block Index
1
file1
2
file1
3
file2
...
file ...

Lecture

- stocker des informations dans un tampon
- avance du curseur de fichier

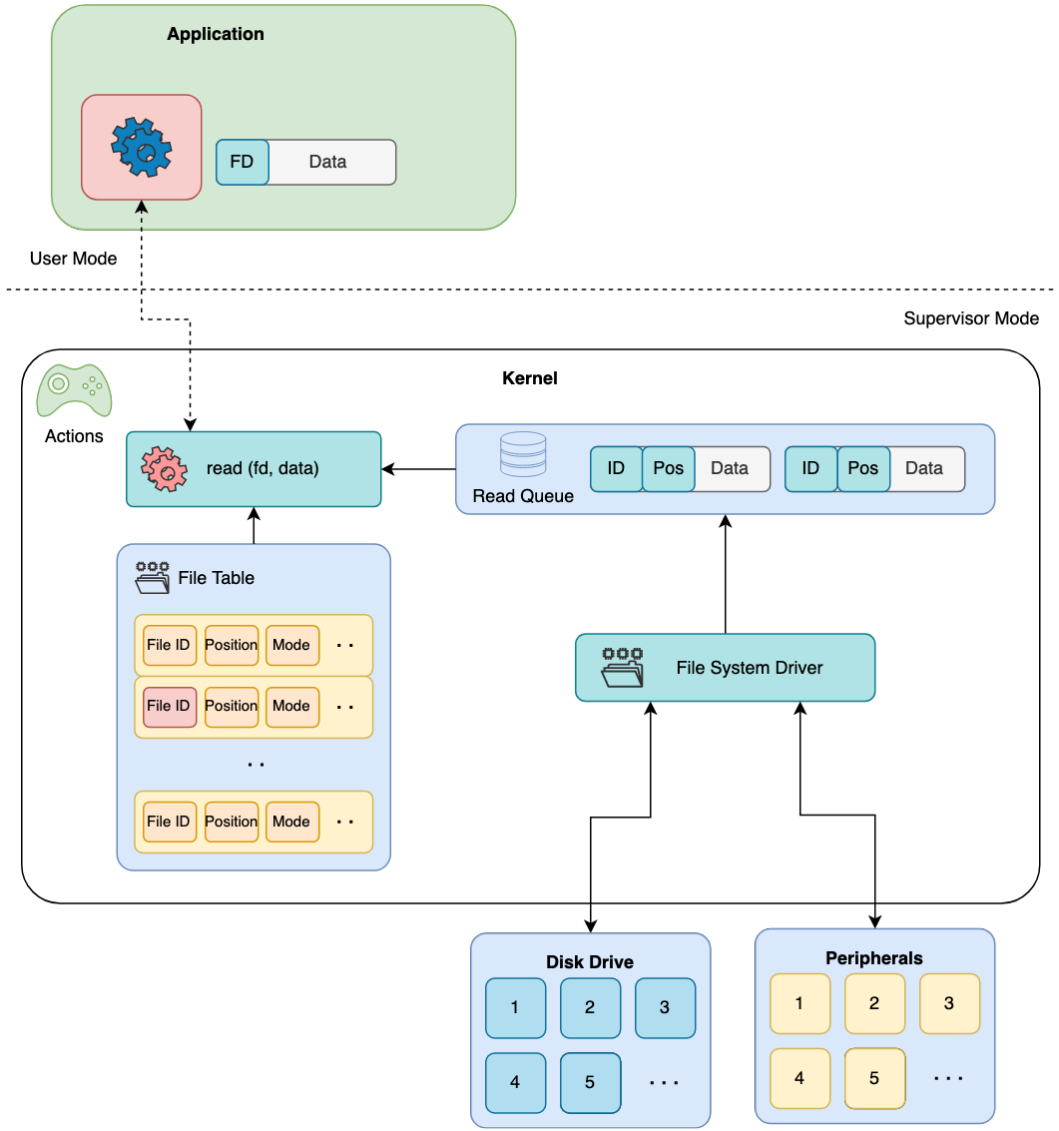
- **Rust**

```
File::read(&mut self, buf: &mut [u8])
    -> Result<usize, Error>
```

- **Unix**

```
// reads maximum size bytes
 // use only within a loop
 n_read = read(fd, buffer, size);
```

Flux de lecture



Ecriture

- écrire des informations à partir d'un tampon
- avance du curseur de fichier

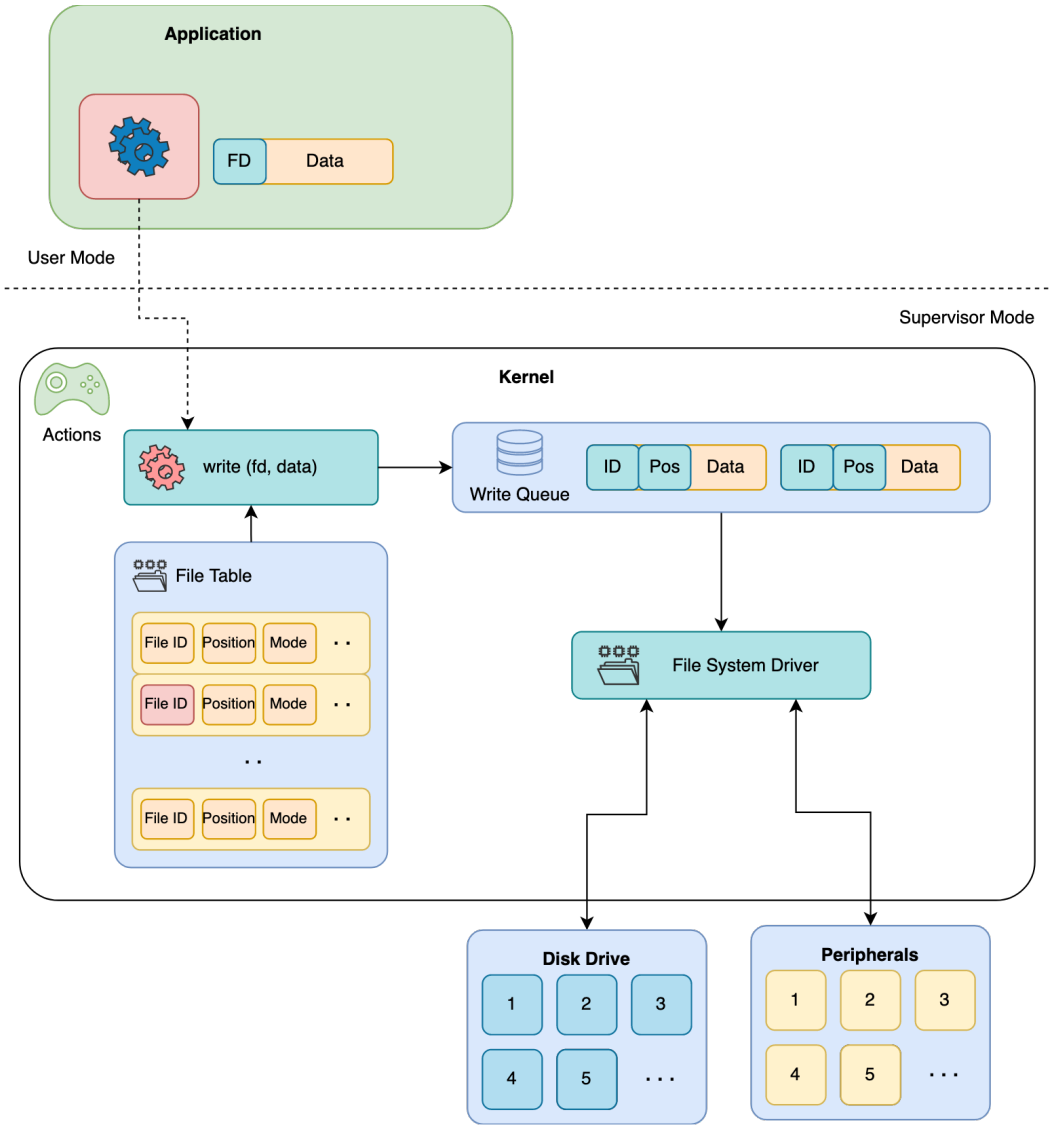
- **Rust**

```
File::write(&mut self, buf: &[u8])
        -> Result<usize, Error>
```

- **Unix**

```
// writes maximum size bytes
  // use only within a loop
  n_write = write(fd, buffer, size);
```

Flux de écriture



Curseur de fichier

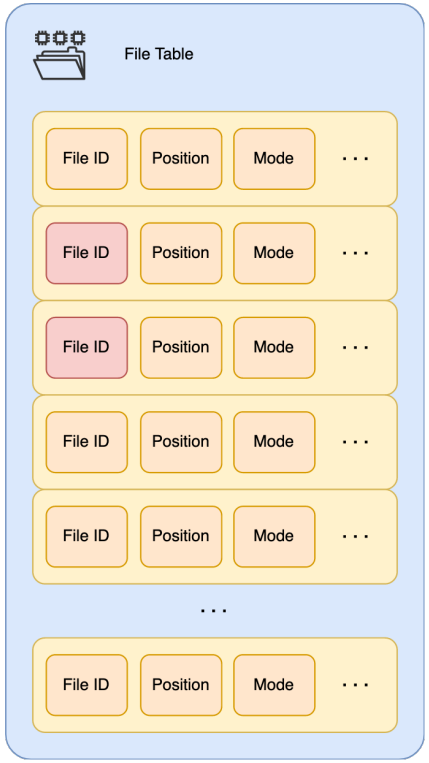
- changements à lire et à écrire
- initialisation à l'ouverture
- mouvement de curseur de fichier

- **Rust**

```
File::seek (&mut self, seek_from: SeekFrom)
    -> Result<u64, Error>
pub enum SeekFrom {
        Start(u64),
        End(i64),
      Current(i64),
}
```

- **Unix**

```
lseek(fd, offset, SEEK_SET);
```



Tronquer

- supprimer le contenu du fichier
- le curseur de fichier est mis à *size*

- **Rust**

```
File::set_len(&self, size: u64)
    -> Result<(), Error>
```

- **POSIX**

```
truncate("/path/to/file", size);
```

Fermer

- l'entrée de la table de descripteur est supprimée

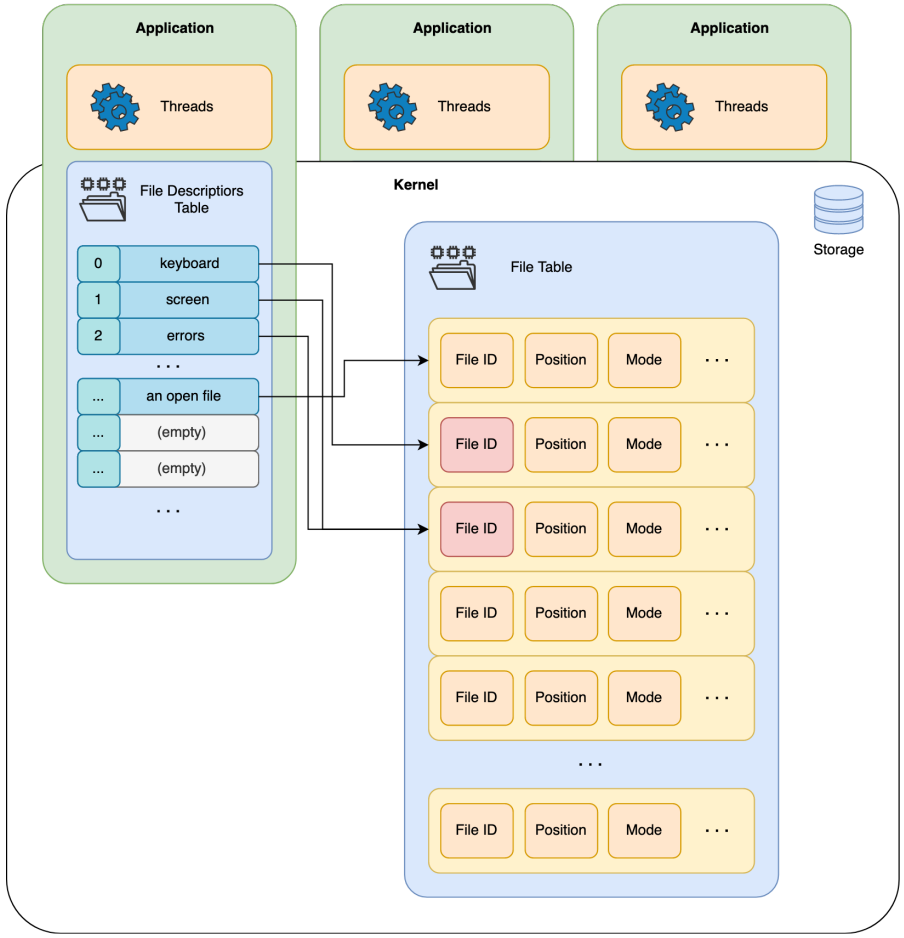
- **Rust**

```
drop(file);
```

- **POSIX**

```
close(fd);
```

Fermer


Application
Threads
Application
Threads
Application
Threads
File Descriptiors Table
0
keyboard
1
screen
2
errors
...
an open file
...
(empty)
...
(empty)
Kernel
Storage
File Table
File ID
Position
Mode
File ID
Position
Mode
File ID
Position
Mode
File ID
Position
Mode
File ID
Position
Mode
File ID
Position
Mode

Types des données

- **Utilisateur**
  - Nom de fichier
- **Application**
  - Descripteur de fichier
- **Système d'Exploitation**
  - table de descripteur
  - structure de données pour le fichier ouverte
  - structure de données pour le fichier sur de disc

Mot clés

- Partition
- Montage
- System de fichiers
- Fichier
- Directeur
- Lien
- Dispositif caractère
- Dispositif bloc
- Socket UNIX
- FIFO
- Permissions
- Chemin de fichier
- Absolue
- Relatif
- MBR
- GPT

Questions